Debugging on Mac OS X

Session 909

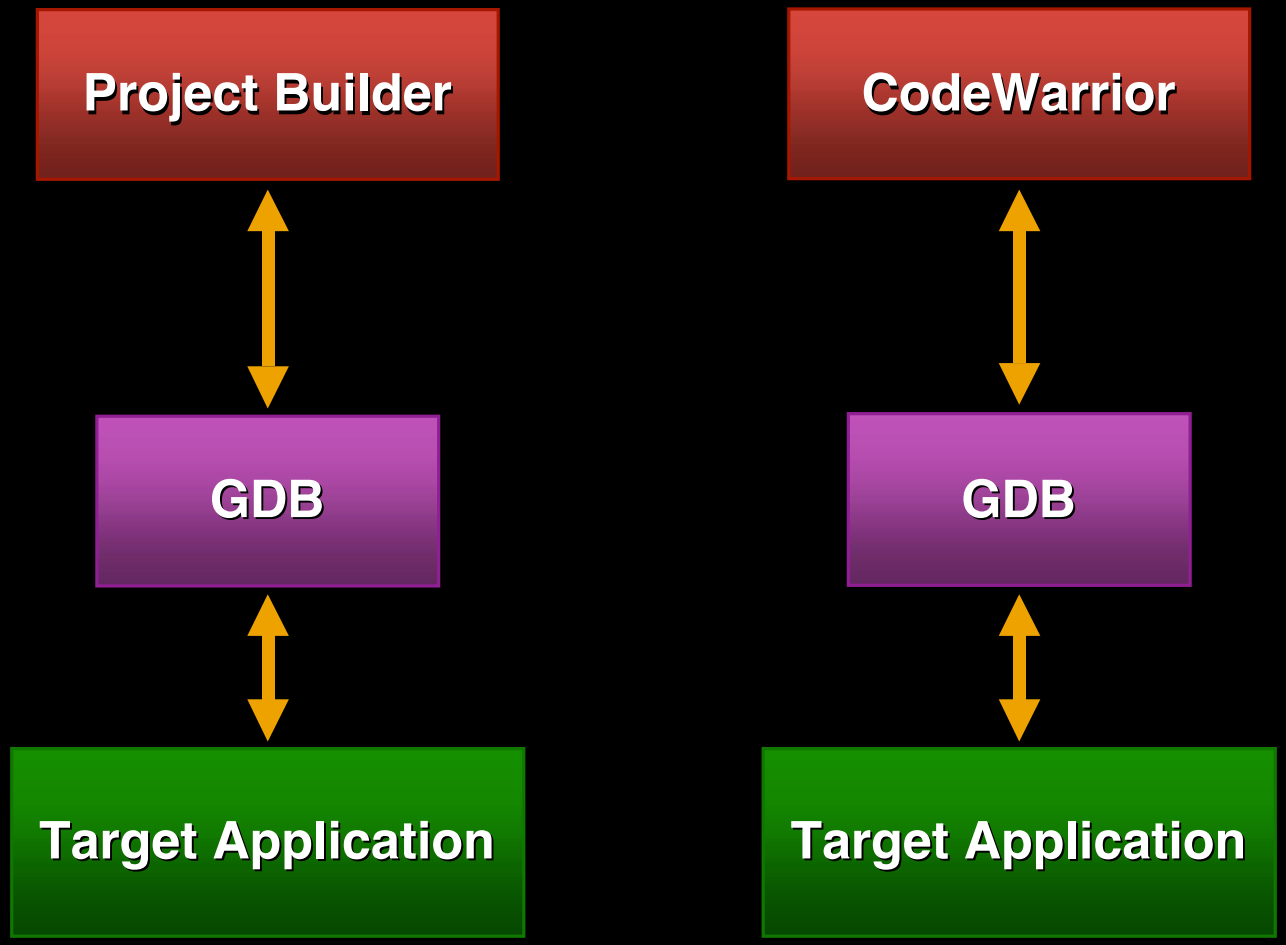
Debugging on Mac OS X

Dave Payne
Manager, Development Environment Group

What You Will Learn

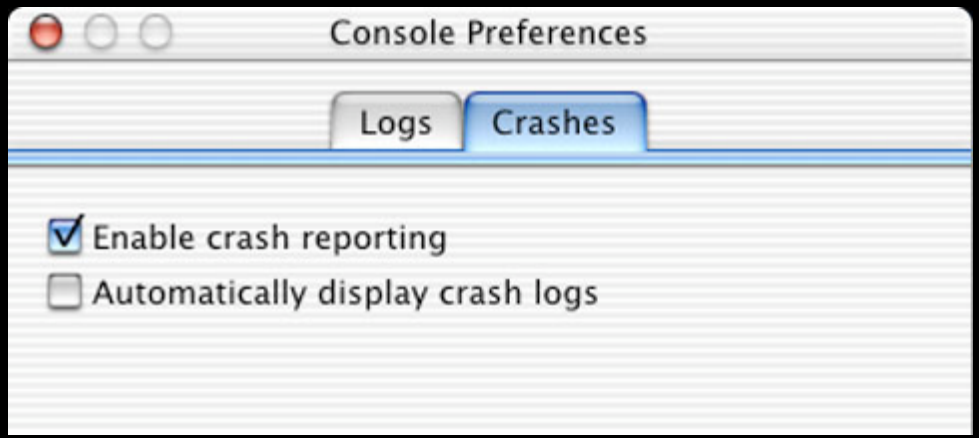
- Approaches to debugging on Mac OS X
- Debugging with Project Builder
- Debugging with CodeWarrior Pro 8
- Advanced debugging with GDB

Mac OS X Debugging Architecture



Why Did My App Crash?

- Turn on crash logging in Console
  - Applications –> Utilities –> Console



Crash Logs in Console

- Crash logs appear in:
  - Separate window for each app in Console
    - Latest crash is at end of window
    - Caution—content retained across reboots!
  - /Library/Logs/<appname>.crashlog
- Logs include:
  - Stack backtrace for all threads
  - Register state for thread that crashed
  - File and line number, if binary has debug symbols

What's Stomping Memory?

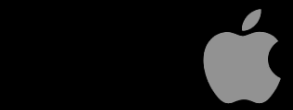
- Hardware watchpoints in gdb
- MallocDebug "helps" detect such problems

Memory Stomper Analysis With MallocDebug

- Encourages badly behaved programs to crash
  - Attach to program with gdb
- Overwrites freed memory
- Adds guard words before and after blocks
- Writes warning messages to Console
- Read documentation in MallocDebug

Why Is My App Hung?

- Use ‘sample’ command line tool
  - Command line version of Sampler
  - Works with any binary
  - Examples:
    - sample <pid> <duration>
    - sample myapp 10
  - Output in /tmp/<app>_<pid>.sample.txt
- ‘gdb attach’



Debugging on Project Builder

Rab Hagy
Project Builder Debugging Engineer

Project Builder Debugging: What You Will Learn

- Project Builder Debugging Overview
- Configuring your project for debugging
  - Build settings
  - Executable settings
- Viewing opaque data types

Project Builder Debugging Overview

- Project Builder IDE Debugger Framework
  - GUI for threads, stacks, variables and PC
  - Breakpoints: persistent, source line, symbolic
  - Cross-project debugging
- Plugins for specific languages
  - AppleScript
  - Java (using wire-protocol)
  - C-based languages (using GDB)

Build for Debugging

- Building Related Projects
  - Use a common build folder for related projects
  - Solves build-time and run-time issues
  - “Project—>Show Info” to set folder
- Compiler Settings For Debugging
  - Enable generation of symbolic information
  - Use Optimization Level Zero (“-O0”)
    - Tip: Verify both Build Style and Target Settings

Executables

- Program which is run or debugged
  - Settings for Arguments, Environment, etc.
- Implicitly created by Application and Tool targets
- Active Target separate from Active Executable
  - E.g. build shared library, run App that uses it
  - Tip: customize toolbar with Active Executable pop-up button

Custom Executables

- External program; not part of your project
  - E.g., program which loads your Plug-in
- Java Tool and Applet targets have default custom executables

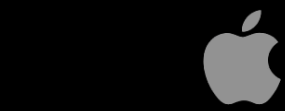
Demo

Executables

David Ewing
Project Builder Engineer

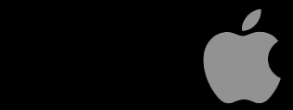
Opaque Data Types

- Types without exposed implementation structure
  - Examples: Carbon, CoreFoundation, Foundation
  - No symbolic information for types
    - “Opaque” even to debugger!
  - Have functional interface
- Use functional interface to types to show content
- First step: NSString
  - Show content and dynamic type of NSStrings
  - Validates ObjC Pointers



Preview: Viewing Data

- Expression Window
  - Add/Remove expression
    - Global
    - Pointer reference
  - Evaluated in UI current context
- Variables in own window



Demo

Data Viewing

Debugging in CodeWarrior Pro 8

Ken Ryall
Director of Debugging, Metrowerks

CodeWarrior Debugging

- CodeWarrior for Mac OS, v8 new debugger features
    - Performance improvements
    - Faster debugging with new linker
    - Smart variable formatting (SVF)
        - Variable formatter
        - Custom data viewers

CodeWarrior Debugging (Cont.)

- More new debugger features
  - Improved memory window and register browser
  - Event points—magic breakpoints
    - Log point
    - Pause point
    - Skip point
    - Script point
    - Sound point

Demo

CodeWarrior on Mac OS X Debugging

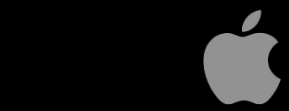
Ken Ryall
Director of Debugging, Metrowerks

Advanced GDB Features

Jim Ingham
Senior Debugging Engineer

Debugging With GDB

- Goals
  - Developers don't need to know GDB
  - GDB console available for advanced users
- Why would you use GDB?
  - Good low-level debugger
  - Situations not yet covered in the GUI
  - Scriptable—looping, conditions, and variables
  - Extensible—user commands, plug-ins
- GDB manual is in:
  /Developer/Documentation/DeveloperTools/gdb/"gdb_toc.html"



Apple Enhancements to GDB

- Objective-C support
- Shared library support
  - ‘future-break’
  - ‘set sharedlibrary load-rules dyld AppKit none’
- Persistent breakpoints (‘save-breakpoints’)
- Plug-in mechanism
  - Write in native code—C / C++ / Objective-C
  - Very fast and flexible
  - Errors in extension will crash GDB
  - Interface will likely change
  - See GPL for redistribution terms
  - MacsBug available in **/usr/libexec/gdb/plugins/MacsBug**

Augmenting Project Builder

- Access to features for which GUI is not (yet) available
  - Remote debugging
  - Watchpoints
  - Kernel debugging

Remote Debugging

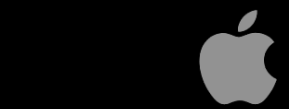
- Advantages
  - Can use debugger without deactivating App
  - Good for debugging full-screen applications
- Steps:
  - Start your application on the remote machine
  - ssh into the remote machine
  - ‘attach’ to application
    - Use AppName<Tab> to look up PID . . .
- Project Builder support planned in the future

Watchpoints

- We now have watchpoint support—
  using page protection
- Fast, unless you are pounding on that page
- Use the GDB “watch” command
- Watch any expression, or give it a raw address

Watchpoints (Cont.)

- Caveats
  - It does not work well for stack objects
  - Because it is a “page protection”scheme, if you watch a page passed into a kernel call, and the call tries to write to the page, the call will fail
  - E.g., watching a buffer that you are passing to “read” will not currently work



Kernel Debugging

- Two-machine solution (KDP protocol)
- Kernel looks like any other program
- Debugging agent in kernel talks to GDB
  - "Hard" kernel lock-ups can hang agent, causing gdb to lose connection to the kernel
  - You can now "detach" from the kernel, and reconnect from another machine
- /Developer/Documentation/Kernel/Tutorials/KEXTutorials/index.html

Advanced Features

- Expression evaluator
- Control constructs
- User-defined commands

Expression Evaluator

- You can call any function in the target
- Functions CHANGE target state
- You can evaluate any expression in the current source language
- You can store values in “convenience variables” which have the form “$name”
- Examples:
  - **call (void) DebugPrintWindowList ()**
  - **set $oldVal = ((Bar *) foo)->someData**

Control Constructs

- You can repeat commands with “while”
  - Syntax:

```
while <Any valid expression>
    <any valid gdb commands>
end
```

- You can do tests with “if”
  - Syntax:

```
if <any valid expression>
    <any gdb commands>
end
```

Example: Linked Lists . . .

- Suppose you have:

```
struct linkedList {
    int someData;
    struct linkedList *next;
} *listHead;
```

- Once you get a long list inspecting the list in the debugger is very tedious
- Can also use this technique for OO collections like NSArray, if there is an iterator method you can call from gdb
    - Harder for C++ STL where you need to create an iterator—you cannot create C++ objects from gdb

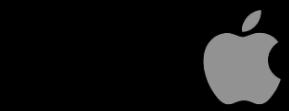
Example: Linked Lists

- Do:

```
(gdb) set $ptr = listHead
(gdb) while $ptr != 0
    printf “Ptr: 0x%lx has value %d\n”, $ptr, $ptr->someData
    set $ptr = $ptr->next
end
Ptr: 0x54a0 has value 9
Ptr: 0x5490 has value 8
Ptr: 0x5480 has value 7
Ptr: 0x5470 has value 6
Ptr: 0x5460 has value 5
Ptr: 0x5450 has value 4
…
```

- Use the “printf” command to make readable output:

```
printf <C format string>, <args>, …
```

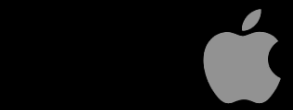


Linked Lists (Cont.)

- Use “if” to hunt out particular elements:

```
(gdb) set $ptr = listHead
(gdb) while $ptr != 0
if $ptr->someData == 5
        printf “Element at 0x%lx = %d\n”, $ptr, $ptr->someData
    end
    set $ptr = $ptr->next
end
Element at 0x5460 = 5
```

- Rummaging through more complex collections works the same way, but you would get the next object with a “next” method of some sort
- Also useful with C Arrays, using an index not a pointer

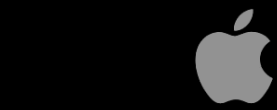


User-Defined Commands

- Complex scripts can be made into user-defined commands:

```
(gdb) define findListElement
    set $ptr = listHead
    while $ptr != 0
        if $ptr->someData == $arg0
            printf ”Element at 0x%lx = %d\n", $ptr, $ptr->someData
        end
        set $ptr = $ptr->next
    end
end
(gdb) findListElement 5
Element at 0x5460 = 5
```

- You can put this in your .gdbinit file, and it will get read in automatically—or you can put it in some other file, and use the gdb “source” command



Summary

- Delivering on our commitment to great development tools
- Powerful debugging support in Project Builder
- Additional improvements on the way
- Please give us your feedback

Technical Documentation

- Command line tools have **man** pages
  - At the terminal prompt: **man <toolname>**
- Reference docs in **/Developer/Documentation/DevTools** for:
  - gdb
  - MachORuntime

For More Information

- Apple Developer Connection tools page
  http://developer.apple.com/tools/
- Project Builder page
  http://developer.apple.com/tools/projectbuilder/
- Apple Developer Connection downloads
  http://connect.apple.com/
- Bug Reporting
  http://developer.apple.com/bugreporter/

Roadmap

San Jose Airport

Who to Contact

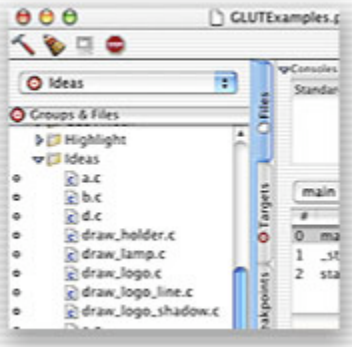
Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

Mailing Lists at Apple
http://lists.apple.com
projectbuilder-users

Development Tools Engineering Feedback
macosx-tools-feedback@group.apple.com

http://developer.apple.com/wwdc2002/urls.html

Q&A



Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002